Point Based Rendering of Implicit 4-Dimensional Surfaces

Ron J. Balsys, Kevin G. Suffern
Faculty of Business and Informatics, Central Queensland University,
Rockhampton M.C., Qld. 4702, Australia
Faculty of Information Technology, University of Technology Sydney,
P.O. Box 123, Broadway NSW 2007, Australia
balsys@cqu.edu.au, kevin@it.uts.edu.au

Abstract

We present a point based rendering algorithm that uses hyper-cubes to perform spatial subdivision in a 4D volume. A 4D function interval exclusion test is used to speed up the rendering of 4D Implicit surfaces in this hyper-volume. A 4D orthonormal basis function is used to define a 4D camera, which projects isometric or perspective views onto a plane in 4D for viewing. The technique requires evaluations of 4D implicit surface functions and gradients to render shaded images. The technique also allows for hidden surface elimination using a z-buffer modified for use in a 4D space. We give examples of its use in rendering some 4D surfaces and discuss problems with the technique. The algorithm can be generalised to higher dimensions.

Keywords—**points, intervals, 16-tree, 4D implicit surfaces**

1 Introduction

There is a large literature on 4D geometry, including the classic books by Abbot [1], Manning [2], and Rucker [3]. A useful web site is hosted by the University of California Irvine [4].

Because this paper is on the rendering of mathematical surfaces in 4D, we cite here some of the relevant literature on this topic. First, there's the book by Coxeter [5], who developed a foundation for geometry in hyperspace. Banchoff has produced a number of works on geometries beyond 3D, including a classic text [6] and a full list of his work can be found on his homepage [7]. Burton $et\ al.$ [8]-[10] was another early contributor to computer graphics techniqiues for rendering in higher dimensional space. A recent thesis by Hollasch [11] discusses ray tracing in 4D. Bhaniramka $et\ al.$ [12] develop an algorithm based on a marching cubes approach for rendering n-dimensional surfaces. Their algorithm renders polygons in the sub-hypercubes of hypercubes.

Relevant work that we have carried out for rendering surfaces in 3D is as follows. Jones $et\ al.$ [13] and Balsys and Suffern [14] presented a point based algorithms for rendering parametric and implicit surfaces and features on surfaces. In common with other point based techniques, in these works we render singular and non-manifold surfaces, and in [14] we render slabs parallel to the principle planes, slabs located along a principal axis and rotated by arbitrary steps, slabs of constant Gaussian and Mean curvatures and produce contour maps of surfaces redered using point primitives.

Balsys $et\ al.$ [15] presented a point based algorithm that uses octree space subdivision with an interval exclusion test to speed up rendering of implicitly defined surfaces in $\Re^3$. Interval arithmetic is discussed in [15]-[21]. Octree adaptive subdivision prior to point sampling greatly speeds up the rendering of implicit surfaces as the nodes converge to the surface. The use of a natural interval exclusion test for the function guarantees that no parts of the surface are missed. This is also discussed in Stolte $et\ al.$[19], and Stolte [20]. In Balsys [15] we also discuss criteria to gaurantee that each plotting node is smaller than its projected image on the image plane so that complete surface coverage by points is assured in 3D.

If a parametric form for the 4D surface exits then parametric methods extended into 4D are a highly efficient method of redering the 4D surface. However a number of surfaces do not have a parametric representation and so other forms of representation must be used. We present here an algorithm for point rendering of shaded implicit surfaces in 4D. An efficient algorithm for rendering a surface would generate a point on the surface each time we generate a sample point as discussed in Balsys $et\ al.$ [15] and here we modify our algorithms in [15] to work in 4D.

For implicit surfaces in $\Re^4$ we replace the cubic node decomposition used in octrees with a hypercube based space subdivision scheme using a 16-tree. This is a tree with 16 children per node. We use the natural interval extension of the surface in $\Re^4$ to exclude from the rendering process, parts of the hypercube that do not contain the

Computer Graphics, Imaging and Visualisation (CGIV 2007)
0-7695-2928-3/07 $25.00 © 2007 IEEE

surface. This means we only generate points in volumes tightly clustered around the surface.

We use an orthonormal basis function to implement a 4D camera, and use the 4D surface function's gradient to shade points on the 4D surface. We illustrate our approach with a number of example surfaces and discuss the strengths and weaknesses of our approach and compare this approach to other existing approaches. We also show how contours can be rendered as slabs in 4D. The algorithms can be generalised to higher dimensions.

2 Point Based Rendering of Implicit Surfaces in 4D

An interval $I = [a, b]$, $a \leq b$ is a set of real numbers defined by $[a, b] = \{x \mid a \leq x \leq b\}$. The $natural\ interval\ extension$ of a rational function $f(x_1, \ldots x_n) : \Re^m \rightarrow \Re$ denoted by $F(X_1, \ldots X_n)$, is obtained by replacing the x_i's by the intervals X_i and evaluating the resultant interval expressions according to the rules of interval arithmetic. See Moore [17], Suffern and Fackerell [16] and Synder [18]. We have implemented interval arithmetic as a $C++$ class. Algorithm 1 details the subdivision where the natural interval extension of the function is used as the subdivision criteria for a hypercube subdivision process.

Algorithm 1. Algorithm for adaptive subdivision of a viewing volume in 4D with an interval exclusion test.

```
void SubDivide (int depth, double x1, double x2,
                double x3, double x4, double d)
{
  for (int j = 0; j ≤ 1; j++)
    for (int k = 0; k ≤ 1; k++)
      for (int l = 0; l ≤ 1; l++)
        for (int m = 0; m ≤ 1; m++)
          CreateNodes (depth+1, x1+j*d/2, x2+k*d/2,
                       x3+l*d/2, x4+m*d/2, d/2);
}

void CreateNodes(int depth, double x1, double x2,
                 double x3, double x4, double d)
{
  int surfacePresent = SurfacePresent (f, x1, x2, x3, x4, d);
  if (!surfacePresent)
    ; // Discard node if surface is not present
  else {
    if (depth == plotDepth) {
      if (surfacePresent) {
        RenderPointsInNode (f, x1, x2, x3, x4, d);
      }
    }
    else
      SubDivide (depth, x1, x2, x3, x4, d);
  }
}
```

Algorithm 1 shows the details of the 16-tree subdivision, and how the interval extension of the function is used to discard nodes that do not contain the surface. The algorithm starts with a call to $CreateNodes()$ with $depth$ set to the maximum plot depth, x_1, x_2, x_3, and x_4's values are set to the centre of the root node in 4D space, and d is the width of the root node.The centre and width define a hypercube that is used as the viewing volume and the function $Subdivide()$ divides this viewing volume into 16 sub-hypercubes (nodes) in 4D. Each node is checked to see if it may contain a section of the surface by calling the function $SurfacePresent()$ which performs the interval exclusion test. If it may contain the surface, the node is recursively subdivided. The recursion proceeds down to a fixed maximum $depth$ in the tree. The efficiency of this algorithm is due to the use of intervals to drive the subdivision down to a depth where the sub-hypercubes are tightly clustered around the neighbourhood of the surface. This means points are generated in a thin shell around the hyper-surface rather than throughout the solid 4D region. In nodes that may contain the surface at the final depth we use a point in the node to render the surface using a 4D shading algorithm.

Algorithm 2. Algorithm that implements an interval exclusion test. $IntFunct$ has 4 spatial parameters. The $in\_interval()$ method tests if the real number 0.0 is in the resultant interval. The interval test is readily extensible to n-dimensions as indicated.

```
int SurfacePresent(functPtr intervalFunct, double x1,
                   double x2, double x3, double x4, double d)
{
  interval A(x1-d/2, x1+d/2), B(x2-d/2, x2+d/2),
           C(x3-d/2, x3+d/2), D(x4-d/2, x4+d/2);
  interval intervalTemp = (*intervalFunct)(A, B, C, D);
  if ( intervalTemp.in_interval(0.0) )
    return 1;
  else
    return 0;
}
```

Algorithm 2 shows the details of the function $SurfacePresent()$. Here, the $interval$ type is a C++ class that implements interval arithmetic and functions as discussed in Moore [17]. Hansen [21] developed infinite intervals as discussed in Suffern and Fackerell [16] and we extend their use into 4D. For arbitrary (non algebraic) mathematical functions, intervals can only provide an exclusion test. Some inefficiency can result in the use of intervals as *false positives* can be retuned where the surface

does not lie in the interval but the interval test still returns 0 as a member of the interval.

The camera system used is similar to the standard $OpenGL$ camera, but extended to higher dimensions. For n-dimensional space we require n orthonormal basis vectors to define a world to camera transformation system. We start with a point, $\mathbf{P_c}$, that specifies the $camera$ position and another point, $\mathbf{P_l}$, that specifies the $look\ at$ point. These two points define a basis vector, $\mathbf{b_1}$ in $n-$dimensional space.

In 3D an *arbitrary up* vector is given to define the sense of *up* along the camera to look at point vector. In 4D these are referred to as *ana* and *kata* as first given by Hinton [22]. In higher dimensions $n-2$ arbitrary vectors must also be specified in such a way. These, together with the camera-look vector, define $n-1$ basis vectors. The final basis vector can be determined as the outer product of vectors $\mathbf{b_1}, \mathbf{b_2}, \ldots \mathbf{b_i}$, see for example D'Zmura $et\ al.$ [23].

In the 4D case we take the basis vector $\mathbf{b_1}$ and arbitrarily define two other basis vectors $\mathbf{b_2}(1,0,0,0)$ and $\mathbf{b_3}(0,1,0,0)$. The outer product of these 3 basis vectors yields the fourth basis vector $\mathbf{b_4}$.

We can always obtain an isometric view of n-dimensional objects by only plotting the first two dimensions on the view plane. For perspective views we can project from our 4D space onto a 2D plane (embedded in the 4D space) as is done in 3D camera systems. This is similar to the approach in Hollasch [11]. For perspective projection we use the following

$$\begin{aligned} x_{temp} &= VPD * \frac{x_1}{x_4} \\ y_{temp} &= VPD * \frac{x_2}{x_4} \\ z_{eye} &= VPD * \frac{x_3}{x_4} \end{aligned} \tag{1}$$

This takes us to a surface in 3D space where the normal 3D to 2D perspective formula is used to find the eye coordinates on the view plane as follows:

$$\begin{aligned} x_{eye} &= VPD * \frac{x_{temp}}{z_{eye}} \\ y_{eye} &= VPD * \frac{y_{temp}}{z_{eye}} \end{aligned} \tag{2}$$

Here (x_{eye}, y_{eye}) is on the 4D view plane, based on the position of the point (x_1, x_2, x_3, x_4) in 4D, and the distance, VPD, from the camera position to the 4D view plane.

We get special cases above due to the arbitrary choice of the two *up* vectors (the *Ana* and *Kata* vectors) in our derivation of basis vectors. In 4D we get 16 such special cases (rather than the 1 in 3D). These occur when the $camera$ and $look\ at$ vector are aligned along any of the principle 4D axes, or the x_1x_2, x_1x_3, x_1x_4, x_2x_3, x_2x_4, x_3x_4 directions. These require us to modify the basis vectors we use to define the orthonormal basis function, to include these special cases.

For shading in 4D, we use the approach of Hanson and Heng [24] who use the Phong [25] model

$$I = I_a k_a + I_i[k_d(\mathbf{L}\cdot\mathbf{N}) + k_s(\mathbf{N}\cdot\mathbf{H})^n] \tag{3}$$

where $\mathbf{L}$ is the 4D light position, $\mathbf{N}$ is the 4D vector normal to the surface at the point, $\mathbf{H}$ is the half angle approximation, and the other terms are all constants as defined for the 3D case.

Our shading primitive is a 4D point, and we can apply the above Phong formula provided we can find the normal to the surface at each point. To define a $4D\ normal$ we require 3 orthonormal 4D basis vectors, or for implicit surfaces we can calculate the normal with the 4D gradient of the function $\frac{\delta}{\delta x_i}\ f, i = 1..4$. The inner product also generalises to 4D.

For planar surfaces there may be a problem as there are $n-2$ orthonormal vectors normal to a 2D plane in a $n-$dimensional space. In our 4D case these orthonormal vectors define another 2D plane and an infinite numer of vectors can be embedded in this plane, all of which are *normal* to our original 2D plane. For example, the 4D hypercube has two axes normal to each 2D face of the 3D faces of the hypercube. These two axes define a plane and all vectors in that plane are normal to the 2D face and thus a unique normal to the 2D face cannot be found. In our work we use the term *2D face* to refer to a 2D bounded planar region in 4D space.

3 Surface Examples

We present here a number of surfaces to illustrate the utility of point based rendering in 4D. The 4D ellipsoid is given by the implicit equation

$$f(x_1,x_2,x_3,x_4) = \frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} + \frac{x_3^2}{c^2} + \frac{x_4^2}{d^2} = 1. \tag{4}$$

Figure 1 shows the 4D ellipsoid $a=3, b=6, c=9, d=12$ viewed from a number of directions. Note that these views all appear like 3D ellipsoids. In Figure 6 we render (x_1, x_2, x_3, x_4) contours on the ellipsoid and to aid the visualisation of the 4D nature of the surface. Figure 2 shows the 4D implicit surface for the function

$$f(x_1,x_2,x_3,x_4) = x_1 + x_2^2 + x_3^3 + x_4^4 = 0. \tag{5}$$

As a further example, the iso-potential surface defined by a series of point charges in 4D is given by

$$f(x,y,z,t) =$$

$$\sum_{j=1}^{n} \frac{q_j}{(x-x_j)^2+(y-y_j)^2+(z-z_j)^2+(t-t_j)^2} - c = 0. \quad (6)$$

Here, (x_j, y_j, z_j, t_j) is the location of the charge, with charge value q_j, and c is the value of the potential surface.

According to Schlafli [26] there are 6 regular Platonic 4D solids, which have from 5 to 600 vertices. Several people have rendered these as wire frame images, for example Smith [27] and Bourke [28]. We can use the above point charges to illustrate the Platonic 4D solids by locating a 4D point charge at each of the vertices of the 4D platonic solid. In Figure 3 we illustrate the resultant surfaces.

Our method can also be used to view surfaces embedded in 4-space. A classic example is the flat torus surface given in Hanson and Heng [24] by the parametric formula $x_1 = \cos u$, $x_2 = \sin u$, $x_3 = \cos v$, $x_4 = \sin v$. Here we can generate points directly on the surface using the parameters u and v and there is no need to do spatial subdivision. Figure 4 shows various views of the 4D flat torus. As an analytic formula for the normal to this surface cannot be determined the torus is flat shaded. The problem in determining the normal in this case is due to a dimensionality issue and is not due to an inability to determine an unique normal as described previously for a plane. The flat torus is not a true 4D surface as can be determined from the formula used to define it. The x_1, x_2 parameters are only dependant on u and independant of v and the reverse holds for x_3, x_4. For a hypersurface in $\Re^4$ there should also be a relationship to a third parameter, say w, with at least one of the parameters x_1, x_2, x_3, x_4.

We can also use this method to render non implicit surfaces in 4D such as the hypercube. Figure 5 (a) we show the outlines of the edges of a hypercube and Figure 5 (b) shows a view of the hypercube rendered with different colours for each of the 2D faces, and with the edges of the 2D faces rendered in black. To allow for shading of the faces of the hypercube it is necessary to specify a *unique* normal for each of the faces of the hypercube so they can be independently shaded. This results in determining 24 normals unique to each face and is a tedious process.

4 Summary and Discussion

We have presented a new algorithm for rendering shaded images of 4D implicit and parametric surfaces using 4D points as the rendering primitive. The implicit surface algorithm is based on the recursive spatial subdivision of a hypercube driven by a 4D natural interval exclusion test. This considerably speeds the rendering process compared with rendering points throughout the viewing hypercube, because the plotting nodes make up only a small fraction of the volume of this hypercube. In addition, the interval test guarantees that no sections of the surface in the hypercube are missed. Because 16 sub-hypercubes are generated in each subdivision, the algorithm is fairly slow. However use of high plot depth results in high resolution images. This approach is generalisable to higher dimensions. Affine arithmetic, see for example Stolfi and de Figueiredo [29], may be used instead of interval arithmetic as this generally has a tighter bound on the function in a plotting node, at a slighter higher numerical cost.

The technique described here has a number of advantages and disadvantages compared to the marching cube polygonisation discussed in Bhaniramka *et al.* [12]. As a high subdivision depth and a large number of points are needed to completely cover the surface (without using splatting) this approach is slower than that for polygonisation. An advantage however is that it can render non-manifold features of surfaces (such as cusps, ridges and curves of self-intersection) that are difficult or cannot be rendered with polygonisation approaches. The speed of performance is probably similar to that achieve in ray tracing 4D surfaces as given in Hollasch [11] but this has not been verified. An advantage over the ray tracing approach of Hollasch [11] is that the ray-surface intersections do not need to be calculated, as a rendered point in the plotting node results from node convergence based on interval analysis. This means that surfaces for which the ray-surface intersection equation cannot be solved can still be rendered by this approach.

The algorithm can also be used to render contours on the 4D surface to aid their visualisation, if we can find an expression for ∇f for the surface. In 4D, $\nabla f = f_x\mathbf{i} + f_y\mathbf{j} + f_z\mathbf{k} + f_w\mathbf{l}$. We show this in Figure 6 for the 4D ellipsoid surface where we render slabs of constant thickness on the 4D ellipsoid. To maintain the slab thickness we modified the formula given in Balsys and Suffern [14] to work in 4D. As this is straight forward we do not give details here.

A number of applications for rendering n-dimensional data are described in the literature. In Bhaniramka *et al.* [12] morphing between two 3D solid endpoints is achieved via a rotation in 4D. Also in that work the surface of a particular hydrogen orbital is rendered. D'Zmura *et al.* [30] developed a novel interface for animations based on the idea of a space-time diagram. In this paper the various frames of a 3D animation give coordinates for x_1, x_2, x_3, with the time axis used for x_4. This results in a 4D surface. Interaction with the surface allows for individual frames to be chosen. The space-time diagram allows for the spread of data in the animation to be visualised.

Another application for n-dimensional surface rendering is in data mining and visualisation. Conventionally, only 3 dependant variables can be simultaneously rendered. With the ability to show surfaces whose shapes de-

pend on more than 3 dependant variables (as is done in this work), exploration of multi-dimensional variation can be achieved. An example is the work of Miller and Gavosto [31].

5 Future Work

A number of issues need attention in this work. The main issue is the speed with which rendering takes place. Rendering times for 4D objects are much longer than that of equivalent 3D objects (for example 3D ellipsoid versus 4D ellipsoid). As this approach is esentially a point sampling approach it will be subject to aliasing issues and these need to be addressed.

Our algorithm can be used to generate a point cloud which is then used as input to a splatting algorithm. Pfister $et\ al.$ [32] introduced the concept of the surface element or $surfel$, developed algorithms to render geometries based on these elements, and used surface splatting for visibilty testing. Similarly, Rusinkiewicz and Levoy [33] developed an algorithm that speeds up rendering of point based objects based on splatting. Zwicker $et\ al.$ [34] used the ellipitical weighted average filter, EWA, of Heckbert [35], to avoid aliasing problems with textures when using splatting to render point clouds. The algorithms in this work can be used to generate a sparse point cloud data set which is then rendered using splatting based approaches with all the benefits and limitations of the splatting approach.

One aspect of the work we must consider is how we should go about the visualisation of higher dimension objects. It is apparent that changing the camera view of the object results in a different cross-sectional shape for the image on the view plane. This is not an error in the display system, but rather this is the nature of the objects we are viewing. Our camera system can do *fly a-rounds* of the object. These images can be made into movies and the resultant change in image shape over time can be used to help shape the minds image of the surface. This has been done, for example, by Hanson and Heng [36]. We can also view stereo pairs of the surface. We do this by defining two 4D lines defining our left and right eye camera views which have an angle of separation of 2^o - 4^o. This is illustrated in Figure 7 for the ellipsoid surface.

6 Acknowledgment

The authors wish to acknowledge the support given for this work by their respective universities.

References

[1] Abbott, E.A, 1884. *Flatland: A Romance of Many Dimensions,* New American Library.

[2] Manning, H.P., 1910. *The Fourth Dimension Simply Explained: A Collection of Essays Selected From Those Submitted in the Scientific American's Prize Competition* available at: etext.lib.virginia.edu/toc/modeng/public/ManFour.html

[3] Rucker, Rudolf, 1977. *Geometry, relativity and the fourth dimension,* New York Dover Pub.

[4] UC Irvine Virtual Reality lab 2004. *The 4D Web Page,* Available at: cvr.uci.edu/dzmura/4D/default.htm

[5] Coxeter, H.S.M., 1973. *Regular Polytopes,* 3^{rd} Ed, New York Dover Pub.

[6] Banchoff, Thomas F. 1990. Beyond the third dimension. Geometry, computer graphics and higher dimensions. Scientific American Library Series. *Scientific American Library, New York*.

[7] Banchoff, Thomas F. Available at: math.brown.edu/~howison/newbanchoff/publications/

[8] Carey, S.A., Burton, R.P., and Campbell, D.M., 1987. *Shades of a Higher Dimension*, Computer Graphics World, X, 10, pp. 93-94.

[9] Donald B. Curtis, D.B., Burton, R.P., and Campbell, D.M., 1987. *An Alternative to Cartesian Graphics*, Computer Graphics World, X, 6, pp. 95-98.

[10] Steiner, K.V., and Burton, R.P., 1987. *Hidden Volume Removal in Four Dimensions*, Computer Graphics World, X, 2, pp. 71-74.

[11] Hollasch, S.R. 1991. *Four-Space Visualisation of 4D Objects,* Available at: stevehollasch.com/thesis/

[12] Bhaniramka, P., Wenger, R., Crawfis, R., 2004. *Isosurface Construction in Any Dimension Using Convex Hulls,* IEEE Visualization & Computer Graphics, **10:2**, pgs 130–141.

[13] Jones, H., Balsys, R.J., Suffern, K.G. 2003. Point Based Rendering of Surfaces With Singularities. *ACM/GRAPHITE2003*, 11-14 February 2003, Melbourne, Australia.

[14] Balsys, R.J., Suffern, K.G., 2004. *Point Based Rendering of Non-Manifold Surfaces With Contours.* ACM/Graphite 2004, 15-19 June, Singapore. pp. 1–8.

[15] Balsys, R.J., Suffern, K.G., Jones, H., 2007. *Point Based Rendering of Non-Manifold Surfaces.* Computer Graphics Forum, Dec. 2007.

[16] Suffern, K.G., and Fackerell, E.D. 1991. Interval methods in computer graphics. *Computers and Graphics*, **15**:331–40.

[17] Moore, R.E. 1966. *Interval analysis.* Prentice-Hall, Englewood Cliffs, NJ.

[18] Synder, J.M. 1992. *Generative modelling for computer graphics and CAD.* Academic Press, Boston, MA.

[19] Stolte, N., Kaufman, A. Parallel Spatial Enumeration of Implicit Surfaces using Interval Arithmetic for Octree Generation and its direct Visualization. *In Implicit Surfaces'98*, 81-87, Seattle, 1998.

[20] Stolte, N. Graphics using Implicit Surfaces with Interval Arithmetic based Recursive Voxelization *in Computer Graphics and Imaging CGIM 2003*, 398-024, Honolulu, 2003.

[21] Hansen, E.R. 1980. Global optimisation using interval analysis the multidimensional case. *Numerische Mathematik,* **34**:247270.

[22] Hinton, C.H. 1884. *Speculations on the Fourth Dimension, Selected Writings of Charles H. Hinton,* Copyright 1980 by Dover Publications, Inc.

[23] D'Zmura, M., Colantoni, P., Seyranian, G., 2001. *Virtual environments with four or more spatial dimensions.* Presence 9, pp. 616–631.

[24] Hanson, A.J., Heng, P.A., 1992. *Illuminating the fourth dimension,* IEEE Computer Graphics and Applications, **12**:4, pp. 54–62.

[25] Phong, B.T., 1975. *Illumination for Computer Generated Pictures,* Graphics and Image Processing, Vol. 18, No. 6, pp. 311–317.

[26] Schlafli, L. 1901. *Theorie der vielfachen Kontinuitat, Auftrage der Denkschriften-Kommission der Schweizer naturforschender Gesellschaft,* Zurcher & Furrer.

[27] Smith, T., 2005. Available at: valdostamuseum.org/hamsmith/24anime.html

[28] Bourke, P., 2005. Available at: astronomy.swin.edu.au/ pbourke/polyhedra/platonic4d/

[29] D'Zmura, M., Colantoni, P., Seyranian, G., 2000. *Visualisation of events from arbitrary spacetime perspectives.* In Erbacher, R.F., Chen, P.C., Roberts, J.C. and Wittenbrink, C.M. (Eds.) Visual Data Exploration and Analysis, VII 3860, pp. 35–40.

[30] Miller, J.R., Gavosto, A.G., 2004. *The Immersive Visualization Probe for Exploring n-Dimensional Spaces.* , IEEE Computer Graphics and Applications, **24**:1, pp. 76–85.

[31] Pfister, H., Zwicker, M., VanBaar, J., and Gross, M. 2000. Surfels: Surface elements as rendering primatives. *In SIGGRAPH 2000*, 335–342.

[32] Rusinkiewicz, S., Levoy, M. 2000. QSplat: A multiresolution point rendering system for large meshes. *In SIGGRAPH 2000*, 343–352.

[33] Zwicker, M., Pfister, H., Van Baar, J., Gross, M. 2001. Surface splatting *In SIGGRAPH 2001*, 371–378.

[34] Comba, J. L. D. and Stolfi, J. 1993. Affine arithmetic and its applications to computer graphics, *Proc. SIBGRAPI'93 VI Simpsio Brasileiro de Computao Grfica e Processamento de Imagens (Recife, BR)*, 9-18.

[35] Heckbert, P. 1989. Fundementals of texture mapping and image warping. *Master's thesis, University of California at Berkeley, Dept. of Elec. Eng. and Computer Science*

[36] Hanson, A.J., and Heng, P.A. 1992. *Foursight.* Siggraph Video Review, 85(11), 4:30 minute.

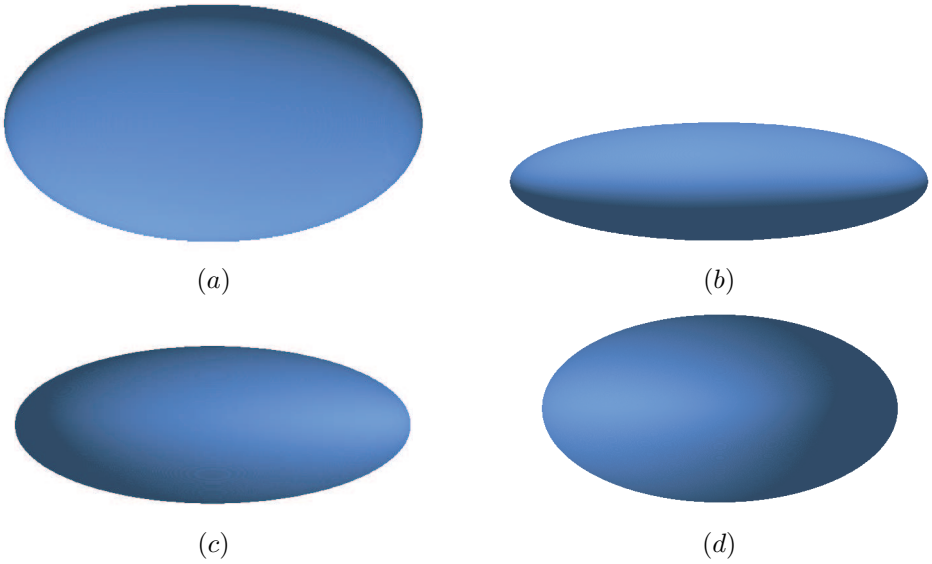


Figure 1: The 4D ellipsoid surface with a=3, b = 6, c = 9 and d = 12. The distance from the camera to the centre of the ellipse is the same in all these figures. (a) viewed along x_1 axis, (b) viewed along x_2 axis, (c) viewed along x_3 axis and (d) viewed along x_4 axis.

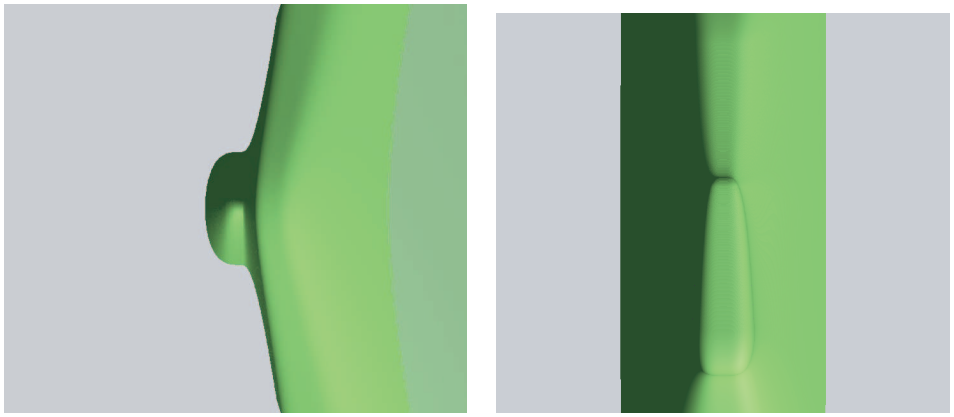


Figure 2: The implicit surface given by $f(x_1, x_2, x_3, x_4) = x_1 + x_2^2 + x_3^3 + x_4^4 = 0$. Viewed along the (a) x_1 and (b) x_3 axes.

Computer Graphics, Imaging and Visualisation (CGIV 2007)
0-7695-2928-3/07 $25.00 © 2007 IEEE

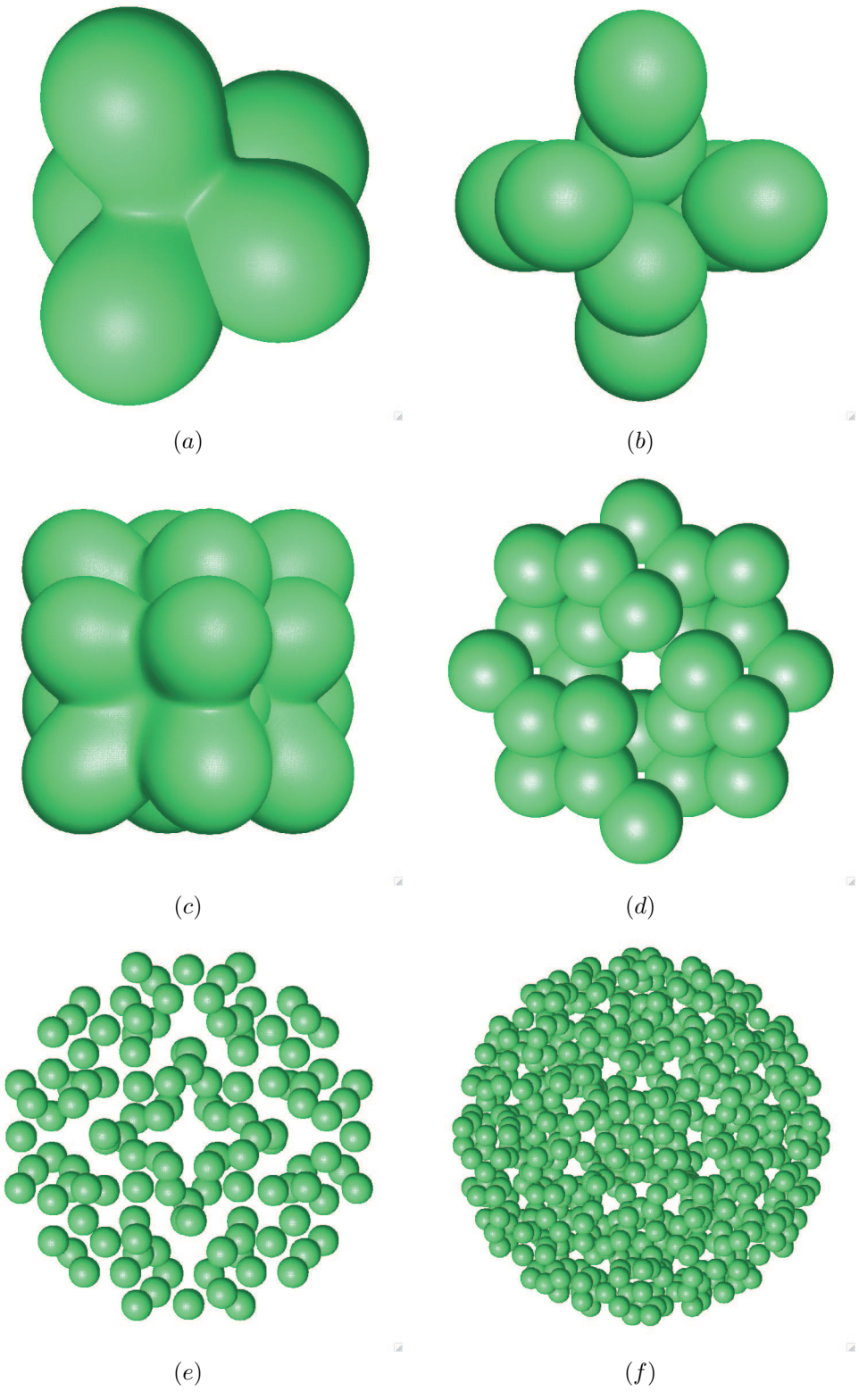
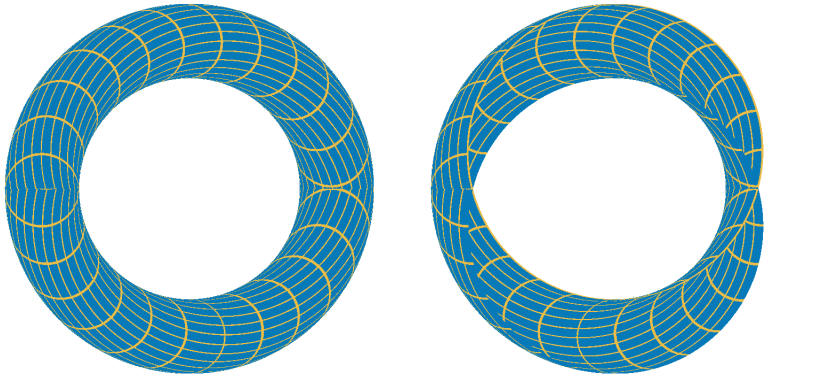


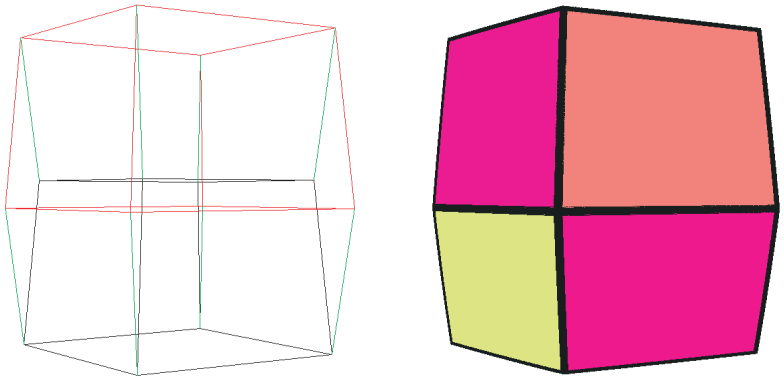
Figure 3: Equipotential surfaces defined by 4D Platonic solids. (a) 4D simplex with 5 charges, (b) surface with 8 charges, (c) surfaces with 16 charges, (d) Surface with 24 charges, (e) surface with 120 charges, (f) surface with 600 charges.

Computer Graphics, Imaging and Visualisation (CGIV 2007)
0-7695-2928-3/07 $25.00 © 2007 IEEE



(a) (b)

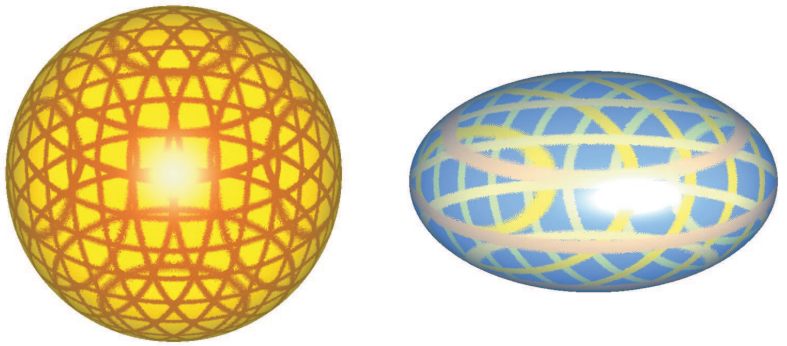
Figure 4: The 4D parametric torus surface.(a) complete surface showing line of discontinuity on surface, (b) clipped $0 < u < \pi$ and $0 < v < 2\pi$ to show u, v contours on the inside and outside of the surface.



(a) (b)

Figure 5: (a) Outline of hypercube. Red lines have $x_1 = 0$, Black lines have $x_4 = 0$. Green lines connect Red to Black vertices. (b) View of the hypercube with each face shaded a different colour.

Computer Graphics, Imaging and Visualisation (CGIV 2007)
0-7695-2928-3/07 $25.00 © 2007 IEEE



(a) (b)

Figure 6: (a) 4D sphere and (b) the 4D ellipsoid surface with a=3, b = 6, $c = 9$ and $d = 12$. Both surfaces are rendered with contours along the x_1, x_2, x_3 and x_4 axes viewed from the (x_1, x_2, x_3, x_4) direction.

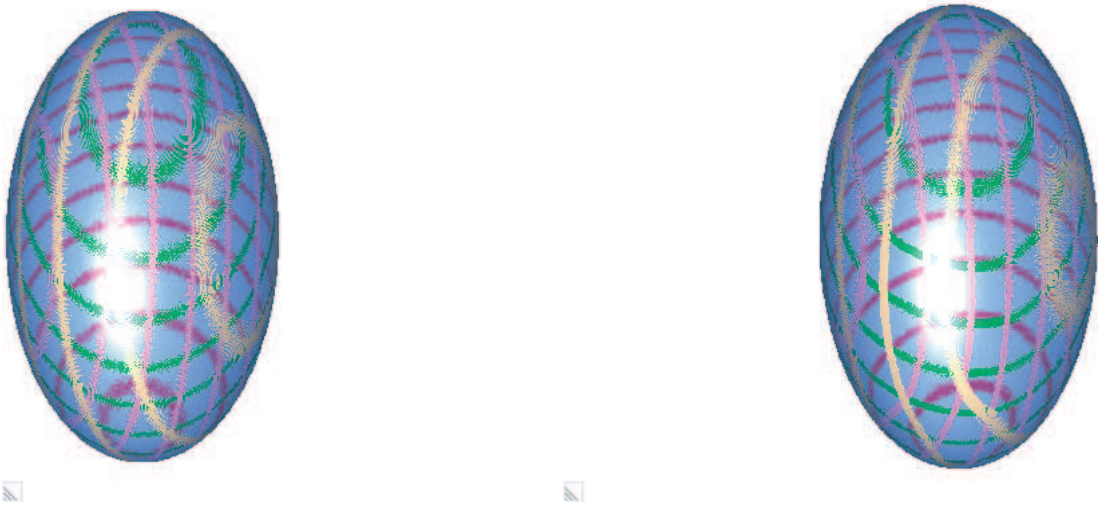


Figure 7: Stereo pair of the 4D ellipsoid surface rendered with contour lines on the surface.